

BACHELOR PROJECT ARCHITECTURE, DESIGN AND IMPLEMENTATION DOCUMENT

BY

DANIEL CHRISTOPHER BIØRRITH, 201909298

LUKAS KOCH VINDBJERG, 201906015

BACHELOR'S THESIS

IN

COMPUTER ENGINEERING

SUPERVISOR: KIM BJERGE

Aarhus University

Department of Electrical and Computer Engineering

June 15, 2022

Glossary

Acronym and technical terms	Description
DFI	Digital Foraminifera Identifier, the formal name of the project, which it will be referred to as throughout the report.
CNC system	Computer numerical control system.
Pylon 6	A software provided by Basler. It comes both with a program, Pylon Viewer, which makes it possible to get livefeed from the microscope, as well as an SDK for developing code to control the camera.
SDK	Software development kit
Foraminifera	A singlecelled organism often found in sediment samples.
Sediment sample	Sediment samples are samples consisting of foraminifera and other granular material.
Tray	The observation platform used to place sediment samples on during scanning.
G-code	A highly popular CNC programming language, used to communicate with the controller.
Grbl	The software located on the controller, which interprets the G-code sent.
VS2013	Visual Studio 2013 is an IDE provided by Microsoft, which is used to develop the acquisition software.
DIP	Digital Image Processing
CV	Computer Vision
GUI	Graphical user interface.
ML	Machine Learning
USB	Universal serial bus
OpenCV	An open source CV library for Python.
Scikit-image	A collection of algorithms for image processing, developed for Python.
Tensorflow	An open source CV library in Python, used for creating ConvNets.
JupyterLab	An IDE primarily used to execute .ipynb files.

Table 0.1: List of glossaries and descriptions thereof

Contents

1	Introduction	3
1.1	Krutchen 4 + 1 View	3
1.1.1	Modification of the model	4
2	Physical View	5
2.1	Deployment diagram	5
3	Process View	8
3.1	Sequence diagram	8
4	Acquisition	10
4.1	Architectural pattern	10
4.2	Logical View	11
4.3	Development View	12
4.3.1	Interface class	13
4.3.2	MessageHandler class	14
4.3.3	ImageGrabber class	16
4.3.4	SimpleSerial class	18
5	Stitching & segmentation	20
5.1	Logical View	20
5.2	Development View	20
5.2.1	ImageProcessor class	23
6	Classification	25
6.1	Logical View	25
6.2	Development View	25
6.2.1	Design choices	26
6.2.2	Training the model	26
6.2.3	Using the model	28
	Bibliography	30

Chapter 1

Introduction

This document aims to give insight into the design and architecture of the DFI system. Additionally, it will describe the implementation that is built upon said designs and architectures. It is a continuation of the requirement documentation[1] where a description of the system, formally called Digital Foraminifera Identifier (DFI), is presented, which is recommended to be read prior to reading this document. The document is built upon the Krutchen 4 + 1 view model[2], developed by Phillip Krutchen, among others by the use of UML diagrams[3]. The implementation will be described by showing and explaining key code snippets.

1.1 Krutchen 4 + 1 View

As mentioned, the system architecture will be documented using Krutchen 4 + 1 view model. The model is used to give multiple perspectives on the system, as to help address design decisions, due to the different interests and problems from each perspective. Additionally, it helps explain the system to the reader, as no single view can explain the working of the whole system. Instead, it helps organize descriptions of the architecture into views to address specific sets of concern from each view. This also helps when describing the system to different stake-holders, as they have their own concerns. The four views are Logical view, Physical view, Development view and Process view. In addition to those views, there is the +1 view in the form of scenarios. It is a generic model that is not restricted to any specific form, which means it can be adapted to a specific design. With this, the developer can chose which specific diagram to utilize at which view. The model is depicted in figure 1.1

The views are briefly explained below, with more info following in their respective section:

- **The logical view:** an object-oriented view. It's concerned with the functionality that the system provides to end-users.
- **The physical view:** also known as the deployment view, visualizes the mapping of the software onto the hardware. Along with this, it shows the inter-connection between hardware components.

- **The development view:** illustrates the system from a programming perspective. It focuses on the actual software module organization in the software development environment
- **The process view:** shows the dynamic aspects of the system. It shows how the different processes communicate in run time.
- **Scenario view:** Shows the architecture by the use of some scenarios or use-cases.

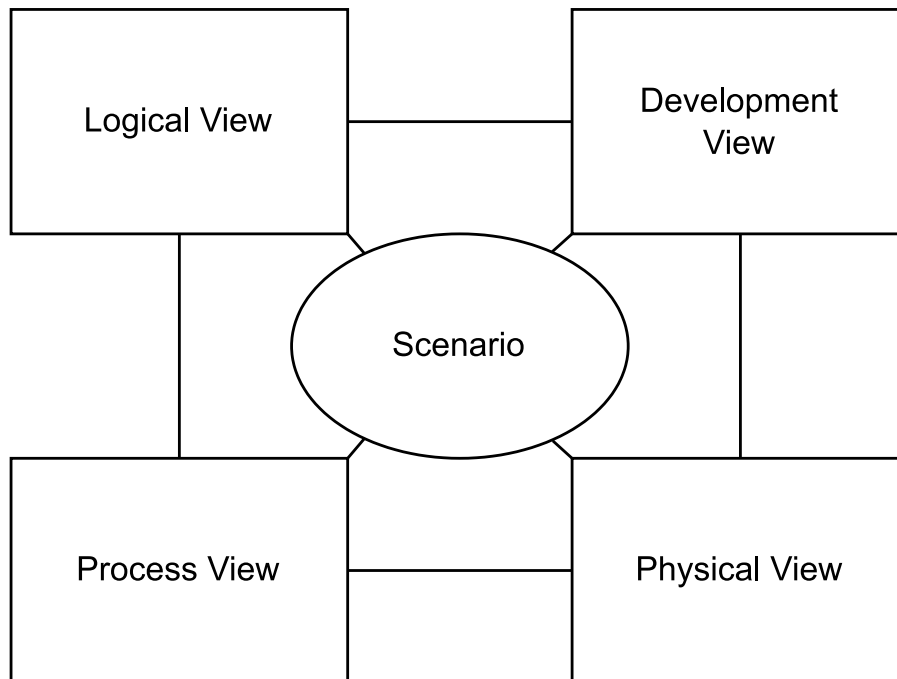


Figure 1.1: Krutchen 4 + 1 view model

1.1.1 Modification of the model

As DFI isn't of particularly large scale, the model would be too extensive to strictly follow, as some information would become redundant. Here, it's advantageous that the model is non-generic. Instead, some custom adjustments/modifications have been made to the model. The process and physical view are not changed, though they are introduced first to provide an abstract description of the system before the other views. Afterwards, the document goes into three sections: acquisition, stitching and segmentation, and classification. Each section has both a logical view and a development view, where the development view will focus on the implementation of the software. The reason for this is they are three individual components independent of each other, by which a continued description through logical and development view for each component is prioritized. The scenario view is left out completely, as it was deemed unnecessary to include for this system to be explained.

Chapter 2

Physical View

This section will present the physical view. The view visualizes the architecture, by showing and explaining the mapping of software elements onto hardware components. In this section, the physical architecture will be explained with the use of a deployment diagram. It is a high-level view of the architecture and will include a description of the objects, as well as the communication between them.

2.1 Deployment diagram

The deployment diagram developed for the system is depicted in figure 2.1. It shows the hardware components and which software is present on them. With this, it provides a perspective of the physical view. Each component is explained in more detail below, in table 2.1.

Component type	Specification	Description
Device	PC	The host machine for the OS software to run on.
OS	Windows 11	The OS located on the host device is specified as Windows 11, as the has been developed and tested on windows 11. It should be compatible with previous Windows versions as Windows is generally backwards compatible. However, this has not been tested and thus can't be guaranteed to work.
Software	Pylon 6	An SDK provided by Basler to communicate with and control the digital camera.

Software	Acquisition	The software responsible for use-case 1 and 2: the communication with the camera and the controller. It communicates with the camera via Pylon libraries, which is compatible with in C++ Visual Studio 2013[REF]. The communication with the controller goes through a serial port. Here, G-Code[4] commands are sent to the controller.
Software	Stitching and segmentation	The software responsible for use-case 3 and 4: stitching images, and segmentation on said stitched image. It gets access to the images from the Acquisition software through the OS system It is developed in Python.
Software	Classification	The software that is responsible for use-case 5: classifying the objects into their respective classes. It is developed in Python.
CNC machine	Genmitsu 3018 PRO	The computer numerical control system (CNC) machine that the motor and controller is attached to.
Controller	Woodpecker CNC Control Board V3.4	The control board is responsible for controlling the CNC machine. It does this per commands received, in the language G-Code. GRBL 1.1f[5] is installed on the controller which is the brain behind controlling the motors based on said commands.
Motor	Nadalan 42 Stepper Motor	The stepper motors handle the movement in the x and y direction, based on commands sent from the controller.
Camera	Camera system	The whole, fully operational system for grabbing images.
Digital Camera	Basler acA2440-35uc	The digital camera is responsible for grabbing the images and transferring them back to the OS system. Is does this by getting commands via a serial port, from which the embedded Basler software located on the camera handles the grabbing [6].
Lens	M Plan APO 5x	The lens magnifies the sediment samples for the camera, such that they are of a sufficient resolution on the images.

Table 2.1: Details of the deployment diagram's components

As the diagram shows, DFI is intended to be run on a single PC connected to a separate camera system and a separate CNC machine. The camera system is a combination of a lens and a digital camera, which connects to and communicates with the PC through a USB port. The CNC machine consists of a controller, three motors (representing the x , y , and z -axis), and a physical structure. The structure

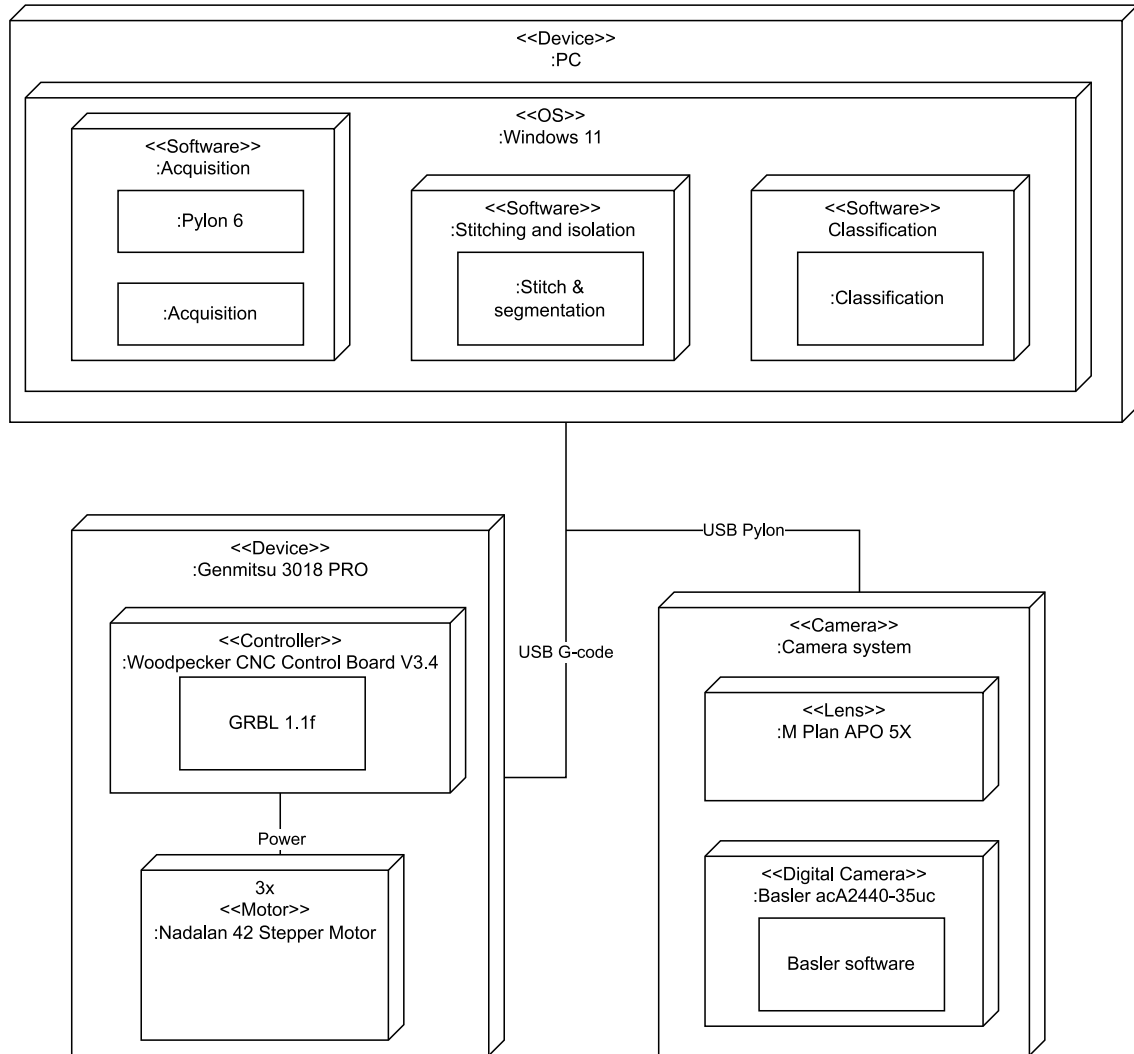


Figure 2.1: Deployment diagram of the full system

is a maneuverable platform, which is able to move the sample relative to the camera. The controller is responsible for the movement by controlling the motors, based on the G-Code commands sent from the PC. There is located three different software packages on the PC: one for use-case 1 and 2 (acquisition), one for use-case 3 and 4 (stitching and segmentation), and one for use-case 5 (classification). The purpose of locating all software packages on the same PC is for easier access to the acquired data and better control flow across the use cases.

Chapter 3

Process View

This section will present the process view. The view shows the dynamic aspects of the system, done with the use of a sequence diagram. This helps show how processes communicate together.

3.1 Sequence diagram

The sequence diagram is depicted in figure [3.1](#). The diagram represents a full walkthrough of all use-cases and is used to show the working of the system. The single actor present is the user of the system (the sediment sample is left out and assumed distributed) and the objects are the software components specified in the physical view, see table [2.1](#).

First, the user starts the acquisition program, from where they have the option to either set the settings of the camera or start the acquisition. Starting the acquisition leads into a loop that finishes when the whole tray has been covered and there are no sediment samples left to scan. Next, with the acquired images, the stitching and segmentation program is to be started. The program first stitches all the images together and afterward performs segmentation on the stitched image. Finally, with the images of the isolated elements, the classification software is started, which executes the classification and finishes.

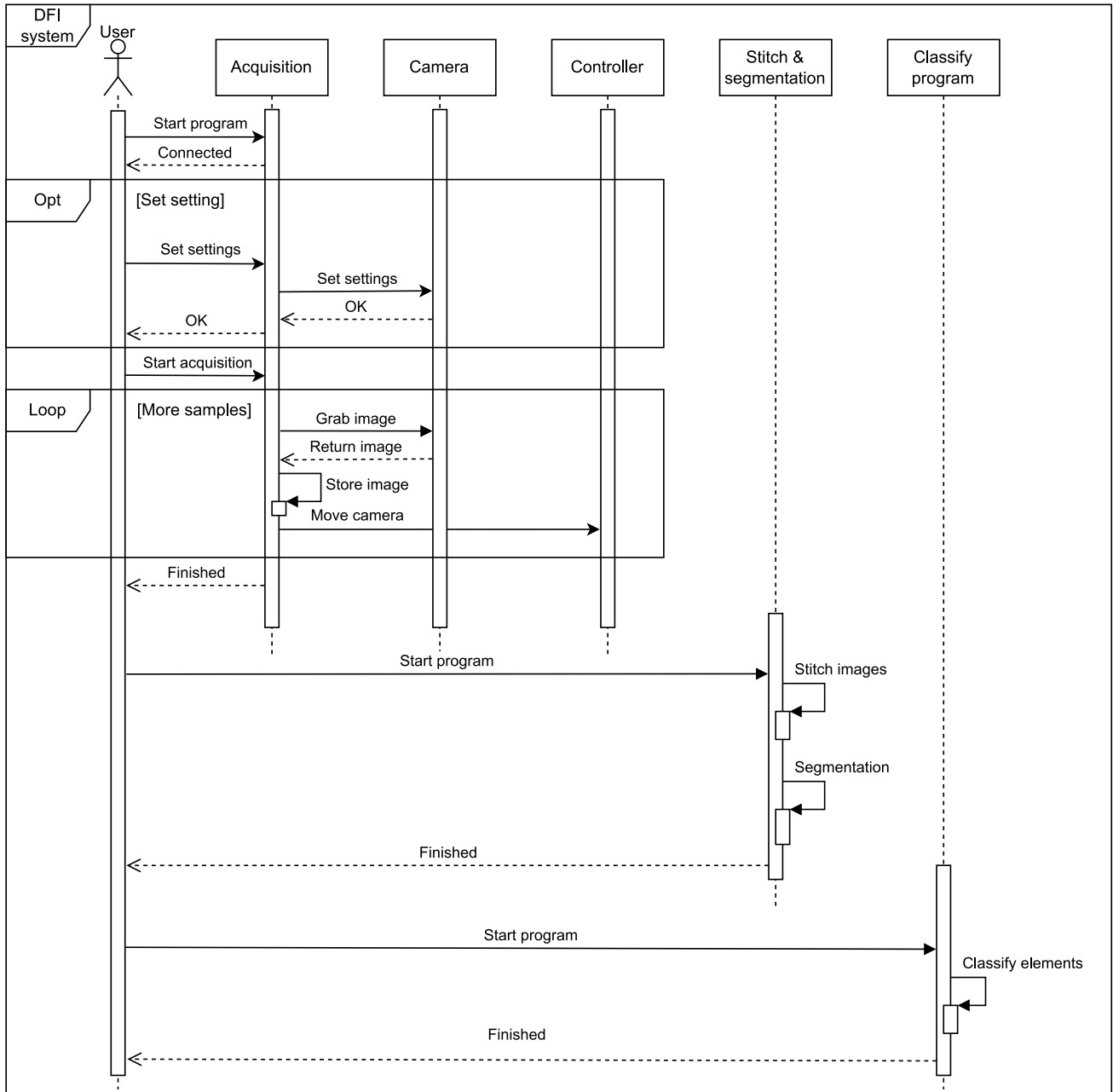


Figure 3.1: Sequence diagram of the full system

Chapter 4

Acquisition

The acquisition software is concerned with satisfying use-case 1 and use-case 2. Specifically, it must provide the user with an interface to change the settings and handle the scanning of the samples. This chapter will provide a description of the acquisition software, by the use of a logical view and a development view.

4.1 Architectural pattern

As the acquisition software needs to both provide the user with an interface, as well as interact with the previously specified hardware, it has been developed following a layered architecture [7, Page 177]. Such an architecture separates the software into different layers, where each layer is dependent on only the layer immediately beneath it.

By keeping the software in different layers changes are localized, meaning they will only affect a subsection of the system. For example, adding or removing a view should not affect other layers. This makes it much easier to modify, add or remove parts of the system, as it only affects the adjacent layer. Our specific layered architecture is depicted in figure 4.1.

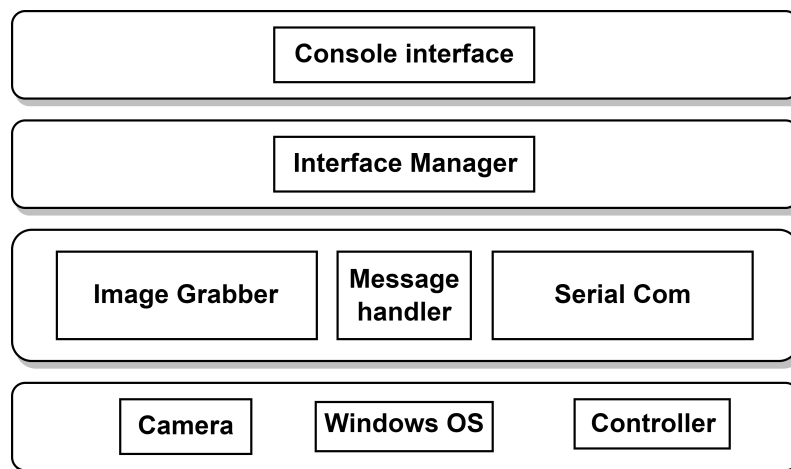


Figure 4.1: The architecture of the acquisition software

The figure shows which elements are present in which layer. In the top layer, you find the simple console interface. Next, the second layer is responsible for handling

said interface, based on requests sent from the user. The third layer is the layer responsible for the gathering of data. It is the layer that sends commands to the camera and controller, prompting the movement of the camera and grabbing of images, as explained in the sequence diagram. Finally, the lowest layer is where the data is grabbed and stored in the Windows OS. With this setup the wanted effect is obtained, as a change in one layer will only affect the adjacent layer. An example of such is a change in the interface manager will only affect the console interface, not the layer responsible for sending commands. This makes future development easier, as, even though it follows the pattern, it is currently not fletched out and could be improved upon for future work.

4.2 Logical View

Next follows the logical view. The view is objective-oriented and will show abstractions of the objects. This will be shown using a class diagram while the implementation of the classes will be shown in the Development View. The class diagram is depicted in figure 4.2. It contains 4 classes: **ImageGrabber**, **SimpleSerial**, **MessageHandler** and **Interface**. The classes will be described below.

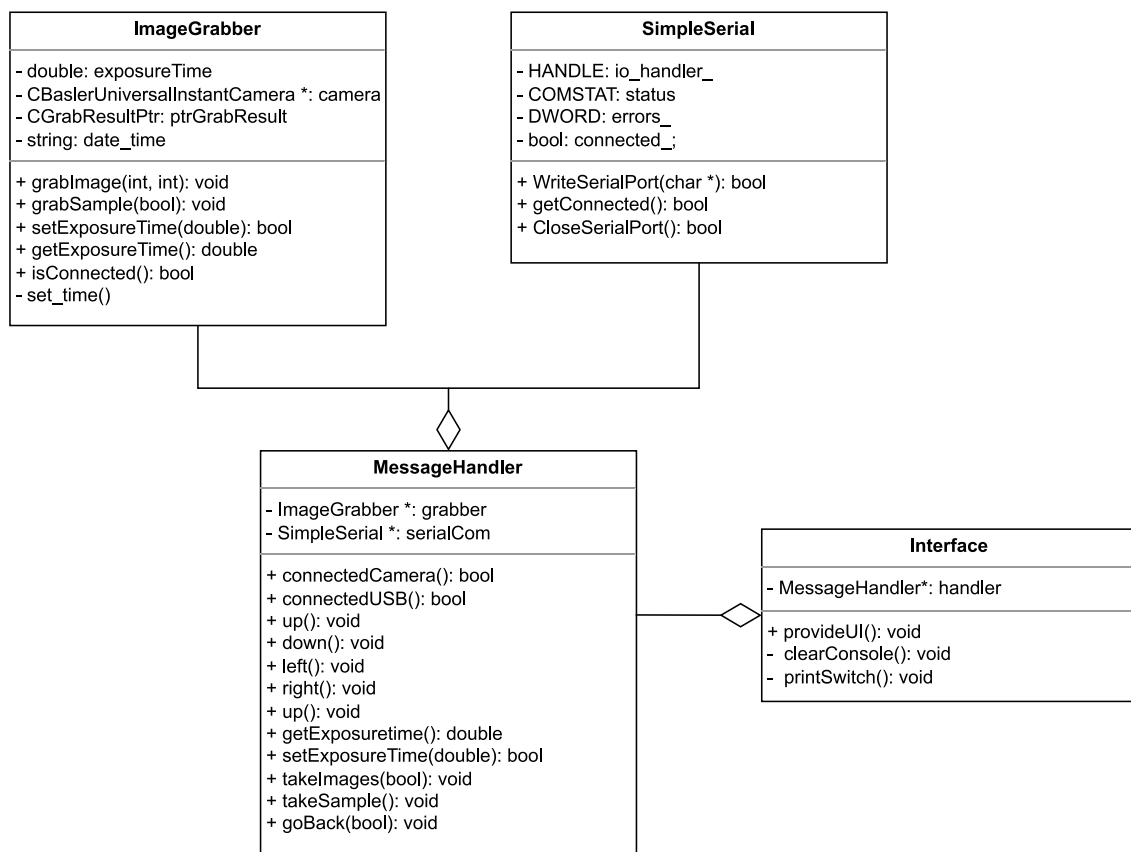


Figure 4.2: Class diagram for the acquisition software

- **Interface**: the class is responsible for managing the user interface of the console. Furthermore, it has an aggregation relationship with the **MessageHandler**,

as it has an instance of it as a private attribute **handler**. The instance is the connection to the next layer.

- **MessageHandler:** the class responsible for sending messages to the **ImageGrabber** and the **SimpleSerial**, based on requests received from the **Interface**. It is an aggregation relationship, between the **MessageHandler** and the two, as it contains instances of them both as private variables, specifically **grabber** and **serialCom**.
- **ImageGrabber:** the class responsible for the grabbing and storing the images by communicating with the camera software and keeping track of the Windows directory images are stored in.
- **SimpleSerial:** the class responsible for the serial connection with the controller located on the CNC machine. It is based on a free to use class found on Github[8].

4.3 Development View

This view shows how the software is decomposed for development, breaking down the software from a programming perspective. It will showcase how the software is developed, both by describing functionalities, but also with the use of key code snippets from the implementation. The whole source code can be found on the DFI Github repository[9].

For reason described in technology research [10], the implementation of the camera system is developed in C++ using Visual Studio 2013, as required to work with Pylon. Consequently, it was decided that the controller software should likewise be developed in C++. Since the camera and CNC router are so closely connected in the final acquisition, implementation them both in C++ greatly simplifies cross communication by only using a single software system.

The system has been developed iteratively, with a focus on first establishing an interface between systems for handling commands, then getting the functionality working and finally providing a thorough UI. During development, decisions about the traversal of the tray and about the storing the images had to be designed.

Regarding the traversing, it is developed with a left-right-left approach depicted in figure 4.3. Furthermore, the distance the camera moves for each picture has a big influence on the behaviour of the system. Moving the camera such that the bottom of one image perfectly matches up with the top of the next image can not be consistently relied on. This becomes especially apparent by again looking at figure 4.3 where the distance traveled between the image at 0x2 and the image at 0x3 is quite far despite the two images being right on top of one another. Any slight drift in each movement of the motors might accumulate. The effect of this becomes especially impactful on the result, as the images are already significantly magnified. Thus, to ensure a consistent result from the process - without any gaps or drifting - some overlap has been introduced for each image. Too much overlap will both increase the time of the process and require more storage as more images would be captured. Thus, a movement of 1 mm for each image was decided on as this seemed

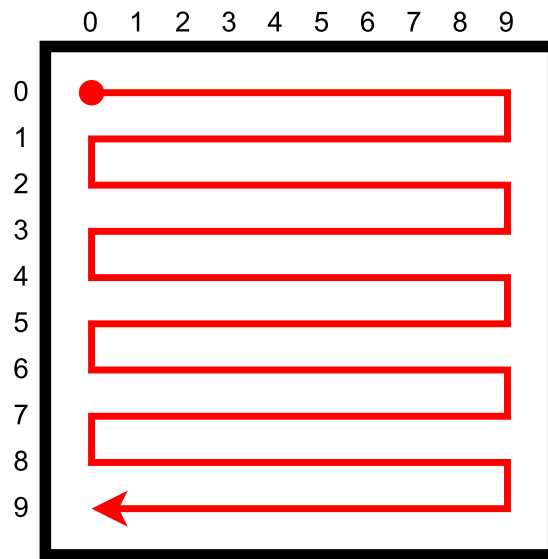


Figure 4.3: The traverse path of the camera.

optimal for each aspect along with it being a simple unit to work with geometrically. The motors will then stop for 0.5 seconds to allow the camera to grab a clear image. Furthermore, the option to only scan a single quadrant was added, as opposed to the whole tray.

The naming scheme of the images are designed to represent their position relative to the starting image. Thus, it follows the 'coordinates' like the matrix naming convention of $N \times M$. This means that the top left image is named '0x0.png' and the bottom right is named '9x9.png' when traversing an area of 10x10 mm., such as in figure 4.3. This both allows for easier assessment of a scanning run and can provide crucial information in later processing steps. Finally, the images are stored in a directory named the current date-time in the format 'Y-m-d H-M-S', which is created when starting the program.

To store the images it was decided not to implement a database. Since the scope of this project is 'proof of concept' it is more interesting to entertain the unique issues that the system has to overcome. Storing images in a database is a much more general task, and thus, does not serve much of a purpose in illustrating the capabilities of the system. Therefore, the images will be stored locally on the user PC with Windows File Explorer[11]. File Explorer provides adequate functionality with a user-friendly UI, and by providing the ability for easy analysis of results.

4.3.1 Interface class

Here follows a description of the `Interface` class and its functions.

Method: `public Interface()`

Parameters: None

Description: a simple constructor which initiates a new `MessageHandler` and stores it in the private `handler` variable.

Method: `private void clearConsole()`

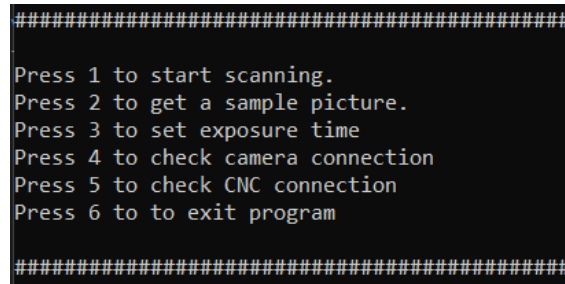
Parameter: None

Description: A function that clears the console.

Method: `public void printSwitch()`

Parameters: None

Description: The function responsible for printing the users options to the console. The console output can be seen in figure 4.4.



```
#####
Press 1 to start scanning.
Press 2 to get a sample picture.
Press 3 to set exposure time
Press 4 to check camera connection
Press 5 to check CNC connection
Press 6 to to exit program
#####
```

Figure 4.4: The console output from the printSwitch function.

Method: `public void provideUI()`.

Parameter: None

Description: A function which contains a large switch statement based on input, from 1 to 6; see listing 1. The sample is a simplified version of the final implementation, yet the core functionality remains. Based on one of the 6 inputs, it handles them in their own respective cases. See comments for explanation of each case.

4.3.2 MessageHandler class

Here follows a description of the `MessageHandler` and its functions. It works as the middleware between the interface and the physical components.

Method: `public MessageHandler()`.

Parameters: None.

Description: a constructor which initiates a new `ImageGrabber` pointer, stored in the private variable `grabber`, and a new `SimpleSerial` stored in the private variable `serialCom`.

Method: `public void up()`

Parameter: None

Description: Of of the functions responsible for the sending G-code. It uses the `serialCom` to send a G-code message to the controller via the serial port. The specific code is `G91G0Y1\n`, prompting it to move 1 in the y direction.

Method: `public void down()`

Parameter: None

Description: Of of the functions responsible for the sending G-code. It uses the `serialCom` to send a G-code message to the controller via the serial port. The spe-

```
1  bool settingUp = true;
2  int scanSpace;
3  while (settingUp) {
4      clearConsole();
5      printSwitch();
6      int input = 0;
7      cin >> input; //Includes a check for invalid input
8      switch (input) {
9          case 1: //Start scan
10             //1 = quadrant, 0 = whole board
11             cin >> scanSpace; //Includes a check for invalid input
12             settingUp = false; //break loop
13             break;
14          case 2: //Get sample
15             handler->takeSample();
16             break;
17          case 3: //Set exposure time
18             double exposureTime;
19             cin >> exposureTime; //Includes a check for invalid input
20             handler->setExposureTime(exposureTime);
21             break;
22          case 4: //Check camera connection
23             handler->connectedCamera();
24             break;
25          case 5: //Check USB connection
26             handler->connectedUSB();
27             break;
28          case 6: //Exit program
29             return;
30          default: //Invalid input
31             break;
32      }
33  }
34  handler->takePictures(scanSpace); //Start scanning when the while loop is broken
```

Listing 1: A dummed down version of the handleUI function. Key functionality remains the same, though without certain safety measures and print statements.

cific code is G91G0Y-1\n, prompting it to move -1 in the y direction.

Method: public void left()

Parameter: None

Description: Of of the functions responsible for the sending G-code. It uses the serialCom to send a G-code message to the controller via the serial port. The specific code is G91G0X1\n, prompting it to move 1 in the x direction.

Method: `public void right()`

Parameter: None

Description: Of the functions responsible for the sending G-code. It uses the `serialCom` to send a G-code message to the controller via the serial port. The specific code is `G91G0X-1\n`, prompting it to move -1 in the x direction.

Method: `public void takeImage (bool)`

Parameter: A `bool` value representing if it is to scan the whole tray or simply a quadrant.

Description: The function responsible for the grabbing of images. It uses the `down`, `left` and `right` functions to move the camera and the `grabber` to take a picture every time it moved.

Method: `public void takeSample()`

Parameter: None

Description: The function that tells the `grabber` to take, and not store, a sample picture.

Method: `public bool connectedUSB()`

Parameter: None

Description: a function that, via the `serialCom`'s function, returns whether the controller is connected or not in the form of a `bool`.

Method: `public bool connectedCamera()`

Parameter: None

Description: a function that, via the `grabber`'s function, returns whether the camera is connected or not in the form of a `bool`.

Method: `public bool setExposureTime(double)`

Parameter: A `double` value being the new, wanted exposure time.

Description: A function that sets the exposure time in the `grabber`, and returns whether it was successfully set or not in the form of a `bool`.

Method: `public double getExposureTime`

Parameter: None

Description: A function that gets the exposure time from the `grabber` and returns it in the form of a `double`.

4.3.3 ImageGrabber class

Here follows a description of the `ImageGrabber` and its functions. It is developed with the use of the Pylon SDK libraries[12][13], by following the samples provided when downloading Pylon 6. A description of each sample file can be found online[14]. Specifically, inspiration was drawn from the files '`grab.cpp`' and '`ParametrizeCamera_UserSets.cpp`', as they show the workings of the `CBaslerUniversalInstantCamera` class and how to parameterize the camera, as

well as how to grab images with the camera.

Method: `public ImageGrabber(double=25000.0)`

Parameter: A double value with a default value of 25000, which can be overwritten in the constructor call.

Description: The constructor first creates the directory in which images will be stored, named the value of the private string variable `date_time`. It uses the standard Windows `CreateDirectory` function for this. Afterwards, it initializes the pylon libraries and declares a new `CBaslerUniversalInstantCamera` stored in the private variable `camera` in a try-catch statement.

Method: `private void set_time()`

Parameter: None

Description: The function that finds the current date-time, in the format 'Y-m-d H-M-S', and store it in the private string variable `date_time`.

Method: `public void grabImage(int, int)`

Parameter: Two integers representing the x and y coordinate for the naming scheme.

Description: The function responsible for grabbing a single image. It does so in a try-catch statement. If an image is grabbed successfully, it displays said image and stores it in the directory created in the constructor, with a naming scheme with a matrix-like representation; 'N×M.png'. A code snippet of the function is shown in listing [2](#).

Method: `public void grabSample(bool)`

Parameter: A bool representing if the image is to be stored or not.

Description: A function much like `grabPicture`, as it grabs a picture. However, this is intended for samples, meaning it either doesn't store the picture or stores it with the name 'sample.png', depending on the input.

Method: `public double getExposureTime()`

Parameter: None

Description: Returns the private value `exposureTime`, which keeps track of the last set exposure time.

Method: `public bool setExposureTime(double)`

Parameter: The double value which the exposure time is to be set as.

Description: Call the camera's '`ExposureTime.SetValue(double)`' function with the provided input in a try-catch statement. If it successfully sets the exposure time, the private value `exposureTime` to the input.

Method: `public bool isConnected()`

Parameter: None

Description: Returns if the camera has been disconnected. If it was never connected in the first place, simply return false.

```
1  try{
2      camera->StartGrabbing(1); //Grab a single picture
3      while (camera->IsGrabbing())
4      {
5          camera->RetrieveResult(5000, ptrGrabResult,
6              ↪ Pylon::TimeoutHandling_ThrowException); //Try to get the image
7          if (ptrGrabResult->GrabSucceeded()) //If success store image
8          {
9              //Save image taken from Utility_ImageLoadAndSave file in the
10             ↪ date_time directory
11             string file_dest = ... //destination name
12             const char *file_dest_char = file_dest.c_str(); //Reformat the image
13             ↪ to be able to store it.
14             CImagePersistence::Save(ImageFileFormat_Png, file_dest_char,
15             ↪ ptrGrabResult);
16             Pylon::DisplayImage(1, ptrGrabResult); // Display the grabbed image.
17         }
18     }
19 }
20 catch (const Pylon::GenericException& e){
21     // Error handling.
22 }
```

Listing 2: A dummed down version of the grabImage function. It shows how an image is grabbed and stored, using the camera class provided by Pylon.

Method: public void goBack()

Parameter: A bool representing if the scanning was of a quadrant or the whole tray.

Description: A function that sends messages in order to move the board back to the original position.

4.3.4 SimpleSerial class

Here follows a description of the SimpleSerial and its functions. The code is, as mentioned previously, taken from a Github project, with minimal to no changes made. Only the functions used in the project will be explained here.

Method: public SimpleSerial(char*, DWORD)

Parameter: The char array (or pointer to string) is the com_port to connect to, while the DWORD is the Baud Rate. Both are used to connect to the controller.

Description: The constructor, responsible for connecting to the serial port provided.

Method: `public bool WriteSerialPort(char*)`

Parameter: The char array (or pointer to string first entry) is the message one wishes to send.

Description: The function that sends a message over the opened serial port. If successful return true, it not return false.

Method: `public bool getConnected()`

Parameter: None

Description: Returns whether the serial port is connected/open or not, by returning a bool.

Method: `public bool CloseSerialPort()`

Parameter: None

Description: The function that closes an opened serial port.

Chapter 5

Stitching & segmentation

This section will provide a description of the stitching and segmentation software. The software is responsible for covering use-case 3 & 4. This means that it will describe the process of stitching the provided images and, moreover, segment the objects from the given image. The description includes a logical view and a development view. To stitch and segment objects, a range of digital image processing (DIP) methods have been used. The general process is to first stitch the pictures together. This step is necessary due to the overlap of images from the acquisition step as an object appearing on multiple images will affect the final data. Afterwards, the foreground objects of the resulting image are highlighted and then finally the objects are isolated and saved as individual, squared images.

5.1 Logical View

Next follows the logical view. The view is objective oriented and will show abstractions of the objects. The view is objective-oriented and will show abstractions of the objects. This will be shown using a class diagram while the implementation of the classes will be shown in the Development View. The class diagram is depicted in figure 5.1. It consists of one class: `ImageProcessor`. A description of the class follows:

ImageProcessor

The class is created with adequate variables and functions such that it is capable of orchestrating the entire process from a given image collection to the final collection of segmented granular material. The class gets provided a directory for the collection of images to be processed and another directory path to be created, where the segmented objects will be stored. Additionally, it can take a conditional parameter in the form of a complete picture. This is so that if the user has an image of the entire sediment sample, the stitching process can be skipped.

5.2 Development View

This view will go into detail about how the stitching and segmentation software has been developed. This includes an overview of what decisions for the design have been made, as well as a description of the implementation.

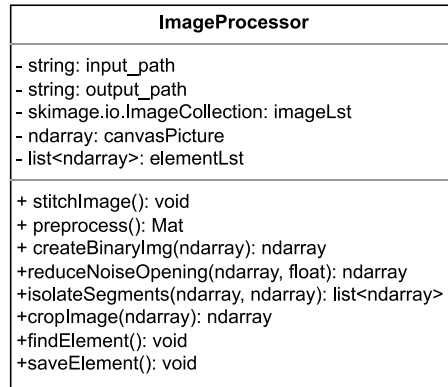
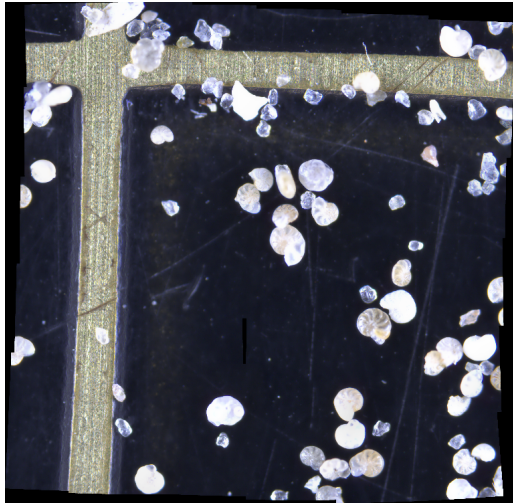


Figure 5.1: Class diagram for the acquisition software

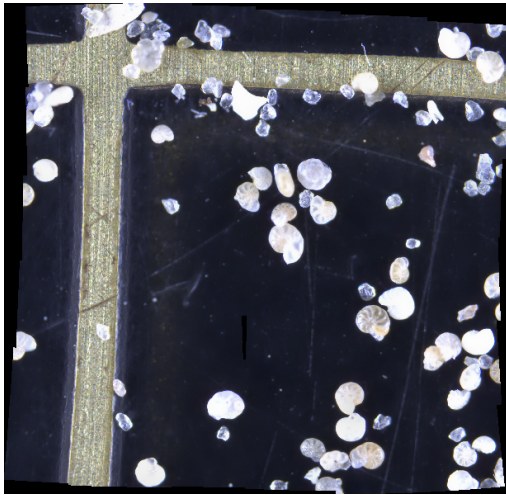
The software goes through multiple processing steps. The first major step is the stitching of images. Stitching is quite a complex process and even requires many advanced processes within. Therefore, OpenCV's image stitching software has been used. However, even though OpenCV is considered a state-of-the-art image processing software, the stitching feature is still quite resource-heavy. The process is both demanding for the processing power required and the needed memory allocation.

It is thus crucial that the collection of images is optimized beforehand to ensure the best results, especially as the software is intended to run on a PC. The optimization can be done both in the acquisition step or with preprocessing of the images. OpenCV's stitching applies the Brown and Lowe method which is capable of dealing with noise and other inconsistencies itself[15]. This means that not much additional preprocessing is required beforehand as the method is robust enough to optimize the images itself. Hence, for a given set of images, the only major parameter to consider for optimization is the scale of the images. Downscaling the images to a lower resolution reduces the processing demands and cuts down on execution time. It also reduces the chance of the process failing due to running out of memory. However, a lower resolution can remove important information from the images which can cause a worse result or even no result at all. The effect of downscaling the images can be seen in figure 5.2. The figure shows the stitching process done of a sample run of a 5x5mm area consisting of 25 images. Here it becomes clear that with a lower resolution, particularly for 5.2c and 5.2d, the process cannot detect a connection between more than just a few images and simply leaves them out. A too low resolution, therefore, yields an unacceptable result. In 5.2a and 5.2b, there is no significant difference in runtime as the images are quite similar¹. As so, the runtime can be completely neglected as a parameter for performance as any stitching less than complete can not be accepted. The desired scale is therefore a scale that will successfully stitch together all of the images but with a scale as low as possible to enable the stitching of a greater number of images without memory error. With the quality and resolution of the images captured in the acquisition stage, it was decided that a downscaling to 40% of the original size was the most optimal value.

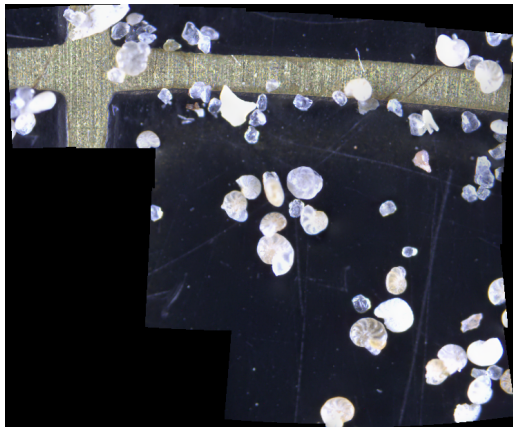
¹For scales of 0.6 and 0.8, the images were likewise complete and with similar runtime which supports the idea that it is not the scale but the number of preserved images that have the biggest effect on runtime



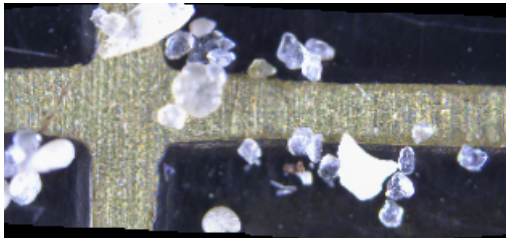
(a) Scale: 1, Avg. runtime: 230.8s



(b) Scale: 0.4, Avg. runtime: 238.1s



(c) Scale: 0.1, Avg. runtime: 10.7s



(d) Scale: 0.07, Avg. runtime: 8.4s

Figure 5.2: Stitching with different scaling of images

Finally, the potential inconvenience of the stitching process is the reasoning behind giving the user the ability to skip this step and provide just a single image if desired.

With the desired image generated, the second part is the segmentation. The first step is to generate a binary image as a foundation for the rest of the processes. The image is first converted to a grayscale image and then to a binary image for a given threshold. This threshold value for foreground pixels can be chosen in many ways but for this system, the choice was either to use the Otsu method or simply decide on a fixed value. The Otsu method can give a different value depending on the provided image. This is useful for images with inconsistent lighting conditions. However, for this system, different acquisition runs - with different amounts and types of granular material - might get a varying threshold. This would in most cases not affect the result as the granular material would likely still be foreground elements. Nevertheless, as the only desired effect is to remove the background, a simple fixed value will be sufficiently reliable as the images are captured under very controlled conditions with the setup of the system. Furthermore, a constant threshold makes the process more optimized and less complex than running the Otsu process.

The binary image will then be cleaned up by using morphological operations.

Here the opening operation was used, to attempt to separate any objects that may be touching. The objects of the binary images were then separated and mapped together with the original coloured image to regain the color of the objects. Finally, the object images were cropped to a square format as this is required for the classification process.

5.2.1 ImageProcessor class

Here follows a description of the `ImageProcessor` and its functions

Method: `def __init__(self, string, string)`

Parameter: The function is provided the input path and the output path for the intended directory of the images collected in the acquisition stage and the desired directory for the segmented objects.

Description: The function is the constructor which gets the relevant information as input and stores it in `self` variables. The variable `imageLst` is created as an array containing all the images and the additional variables `elementLst`, `canvasImg` are created without any values as they will be used later.

Method: `preprocess(self, ndarray, float)`

Parameter: An image is passed as an ndarray and a floating point value to resize.

Description: The function takes the provided image and a value for which it rescales the image

Method: `stitchPictures(self)`

Parameter: Self to access the class variables.

Description: The function takes the images stored in the `imageLst` and calls the `preprocess()`-function to get a list of lower resolution images. These are then stitched together and the resulting image is saved in the variable `canvasImg`.

Method: `createBinaryImg(self, ndarray)`

Parameter: The ndarray is the image which the function should process.

Description: The function takes the given image and creates a binary image consisting of foreground and background pixels only.

Method: `reduceNoiseOpening(self, ndarray, int)`

Parameter: The ndarray is the desired image and the integer is the size of the structuring element by which the process is executed.

Description: The function applies the morphological operations, erosion and dilation, to remove noise from the image and to separate objects that slightly overlap

Method: `isolateSegments(self, ndarray, ndarray)`

Parameter: The ndarray is the image for the segmentation and the ndarray is the structuring element for operations.

Description: The function finds all the grouped sections of foreground elements and creates an returns a `binaryElementLst` which is an array where each image consists of just one object as the foreground element.

Method: `cropElement(self, ndarray)`

Parameter: The ndarray is the image to be processed.

Description: The function takes an image and crops the top, bottom and sides so that there is no excess black pixels. It then adds back some background pixels in the shortest dimension so that the image becomes square.

Method: `findElements(self)`

Parameter: Self to access the class variables.

Description: This function is the main function to generate the list of objects. It calls the `preprocessing()`, `createBinaryImg()`, `reduceNoiseOpening()`, `isolateSegments()` and `cropElement()` functions in order with appropriate values.

Method: `saveElements(self)`

Parameter: Self to access the class variables.

Description: The function saves all the elements from `elementLst` to the directory specified in the constructor.

Chapter 6

Classification

The classification software is concerned with satisfying use-case 5, meaning it must classify provided object as either Foraminifera or sand. The following section will provide a description of how it does so, in parts with a logical view and a development view. The whole implementation can be found on Github[9].

To classify the images, convolutional neural network (ConvNet)[16] models were trained and exported to .H5 files. With the H5 files, you can provide inputs to the program, from which it gives a prediction on the type of object. It assigns each possible output class a value in the interval $]0;1[$, where the value represents a probability of the output being the given class. Naturally, it will be classified as the greatest output value. When an object is classified, the image is stored with the type as its name, either 'Foraminifera.png' or 'Sand.png', in their respective folders created upon starting the program.

The model and program is developed in a Python on a Anaconda environment in Visual Studio Code. It was trained on data from the previous FRIDA[17] bachelor project, as they had an exists data set to train the models with.

6.1 Logical View

The logical view is objective-oriented and will show abstractions of the objects. This will be shown using a class diagram, while the implementation will be shown in the Development View.

The class diagram is depicted in figure 6.1. It only contains the class responsible for the classification, as training the model is not built as a class. The class present in the diagram is the **Classifier**; a description of the class follows:

Classifier

The classifier is a simple class which takes a path and classifies all PNG files inside as either Foraminifera or sand. It does so with a H5 file.

6.2 Development View

This view shows how the software is decomposed for development, breaking down the software from a programming perspective. It is split up in two parts; first part

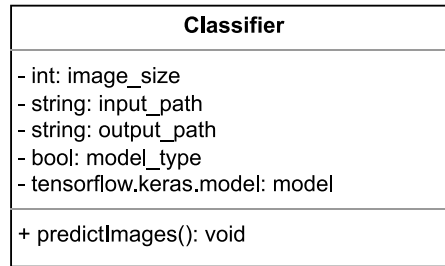
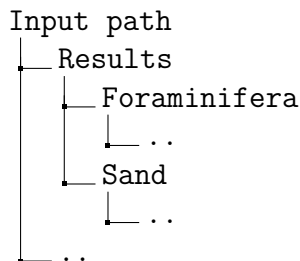


Figure 6.1: Class diagram for the classification software

involves the training of the model, while the second part is the system which uses said model, in the form of the Classifier class.

6.2.1 Design choices

It was decided to store all the classified elements as images inside the provided input path. This would be done with a structure like the one seen below:



Here, all images classified as Foraminifera are stored in the 'Foraminifera' directory, and all classified as sand in the 'Sand' directory. No GUI was created, and the main function has to be modified to change the input path.

6.2.2 Training the model

As stated previously, a convolutional neural network (ConvNet) model was trained for the purpose of classification[18]. The model was an EfficientNetB7 model, trained on the FRIDA dataset[17].

The set was 672 images in total, split out across four classes having 168 images each. The decision to train the model on 90 % of the images (608) was made, instead of a standard 80/20 split, as the dataset is quite small. Additionally, it was expected to validate the model on data collected in this project, meaning the size of the validation set could be decreased.

An 80/20 split was made on the 608 training images, as an error occurred when trying a 90/10 split, for which no solution was found [19]. The batch size was set to 32, providing an epoch length of 15, and it was trained on 58 epochs. The dropout rate was set to 20%, while the learning rate was 0.01 %. The input size was 224 x 224. The summary of the model can be seen in figure 6.2.

All of the parameters are quite standard for training a ConvNet.

The data was loaded in using the Tensorflow `ImageDataGenerator` class, which is done in a standard way found in multiple online guides [20] [18]. The class makes it very easy to have data augmentation, providing randomness in the samples to further improve the model.

Layer (type)	Output Shape	Param #
efficientnetb7 (Functional)	(None, 7, 7, 2560)	64097687
gap (GlobalMaxPooling2D)	(None, 2560)	0
dropout_out (Dropout)	(None, 2560)	0
fc_out (Dense)	(None, 2)	5122
=====		
Total params: 64,102,809		
Trainable params: 5,122		
Non-trainable params: 64,097,687		

Figure 6.2: Summary of the created model.

In code snippet 3, it's shown how the class was created and how the test data was loaded in. Notably, no normalizing was done as it was not necessary. The rotation_range is set to 180, meaning images are rotated randomly between 0 and 180 degrees. Horizontal and vertical flip makes it so random flips occur on the data. The zoom range is random zoom on the images of up to 30 %. The brightness sets a random brightness on the images between 90 and 110%. When the ImageDataGenerator is created, it can be used to load in data from a provided directory. A list of options can be found in [21].

```

1 train_datagen = ImageDataGenerator(
2     rotation_range = 180,
3     horizontal_flip = True,
4     vertical_flip = True,
5     zoom_range=0.3,
6     validation_split=0.2,
7     brightness_range=[0.9, 1.1]
8 )
9 train_generator = train_datagen.flow_from_directory(
10     data_dir, target_directory
11     target_size=(image_size, image_size),
12     batch_size=batch_size,
13     class_mode="categorical",
14     subset='training',
15     shuffle=True,
16     seed=seed
17 )

```

Listing 3: Loading in the training data

The model was created using the tensorflow.keras.applications.EfficientNetB7 with some additional layers added. In code snippet 4, its shown how it was cre-

ated. Apart from the tensorflow base, a maxpooling layer, a dropout layer and a Sigmoid output layer was added, following the example set in [18].

```
1 conv_base = EfficientNetB7(weights="imagenet", include_top=False,
  ↳ input_shape=input_shape)
2 conv_base.trainable = trainable
3 model = models.Sequential()
4 model.add(conv_base)
5 model.add(layers.GlobalMaxPooling2D(name="gap"))
6 if dropout_rate > 0:
7     model.add(layers.Dropout(dropout_rate, name="dropout_out"))
8 model.add(layers.Dense(number_of_classes, activation="sigmoid", name="fc_out"))
9
10 model.compile(
11     #optimizer=optimizers.RMSprop(lr=2e-5), a
12     optimizer=optimizers.Adam(learning_rate=0.0001),
13     loss="categorical_crossentropy",
14     metrics=["acc"],
15 )
```

Listing 4: Architecture and specifications of the developed model

Finally, the data was fit onto the model, as shown in code snippet 5. The steps per epoch depend on the number of training data and the batch size, while the epochs were said to 32 as stated.

```
1 history = model.fit(
2     train_generator,
3     steps_per_epoch= NUM_TRAIN // batch_size,
4     epochs=epochs,
5     validation_data=validation_generator,
6     validation_steps=NUM_TEST // batch_size,
7     verbose=1,
8     callbacks=[
9         tf.keras.callbacks.TensorBoard(log_dir),
10        hp.KerasCallback(log_dir, hparams),
11        ],
12 )
```

Listing 5: Providing the model with the loaded data.

With all this, a model was trained and ready to use.

6.2.3 Using the model

As mentioned, the `Classifier` class uses the model to classify a provided input of images. Here follows a description of the functions, with belonging code snippets.

Method: `__init__(self, string, int, string, bool, string)`

Parameter: The inputs are the path of the H5 model, the image size, the path of the input pictures, the type of the model (bool as we have 2 models) and where the output path.

Description: The function gets the relevant information as input and stores it in `self` variables. It has default values as input, which can be overwritten when calling the constructor.

Method: `classifyImages(self)`

Parameter: Self to access the class variables.

Description: The function that predicts on the images. A code snippet of it can be seen in listing 6. First, It loads in the images in the input path and resizes them. Afterwards, it predicts on the images with the provided model. Based on the maximum index of the prediction result, it either stores the images in the Foraminifera or the Sand directory.

```
1  imgs = [cv2.cvtColor(cv2.imread(file), cv2.COLOR_RGB2BGR) for file in
↪  glob.glob(self.input_path + "/*.png")] #Load in all images
2  i, j = 0, 0 #For naming convention
3  for img in imgs:
4      img = cv2.resize(img, (self.image_size, self.image_size)) #Resize image
5      x = np.expand_dims(img,axis=0)
6      y = self.model.predict(x) #Predict
7      max_index = np.argmax(y[0]) #Find the max index
8      img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR) #Back to RGB
9
10     if(max_index == 3): #3 means sand
11         #Store image
12         cv2.imwrite(self.output_path + "/Sand/sand" + str(i) + ".png", img)
13         i += 1
14     else: #Else its foraminifera
15         cv2.imwrite(self.output_path + "/Foreminifera/foraminifera" + str(j) +
↪         ".png", img) #Store image
16         j += 1
```

Listing 6: Using the model to classify all images in a given directory. First load in the images, then predict on them in a for-loop. Store them according to max index.

Bibliography

- [1] L. K. Vindbjerg and D. C. Biørrieth, “Requirement & testing documentation,” 2022. [Online]. Available: <https://github.com/Biorrieth/Bachelor-project/tree/main/References>.
- [2] P. Kruchten. “Architectural blueprints—the “4+1” view model of software architecture.” (1995), [Online]. Available: <https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf> (visited on 06/09/2022).
- [3] “Unified modelling language.” (2022), [Online]. Available: <https://www.uml-diagrams.org/> (visited on 06/09/2022).
- [4] rebrap. “G-code.” (2022), [Online]. Available: <https://rebrap.org/wiki/G-code> (visited on 04/04/2022).
- [5] Grbl CNC controller. “Grbl.” (2021), [Online]. Available: <https://github.com/grbl/grbl> (visited on 04/04/2022).
- [6] Basler AG. “Aca2440-35uc - basler ace.” (2022), [Online]. Available: <https://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace/aca2440-35uc/> (visited on 06/09/2022).
- [7] I. Sommerville, *Software Engineering, Global Edition*, 10. edition. Pearson Education Limited, 2016, ISBN: 9781292096148. [Online]. Available: https://books.google.dk/books?id=W_LjCwAAQBAJ (visited on 03/29/2022).
- [8] D. David Michalik. “Simple serial communication between a windows c++ application and an arduino.” (2022), [Online]. Available: https://github.com/dmicha16/simple_serial_port (visited on 06/09/2022).
- [9] D. C. Biørrieth and L. K. Vindbjerg. “Github repository - bachelor project.” (2022), [Online]. Available: <https://github.com/Biorrieth/Bachelor-project> (visited on 06/07/2022).
- [10] L. K. Vindbjerg and D. C. Biørrieth, “Final documentation,” 2022. [Online]. Available: <https://github.com/Biorrieth/Bachelor-project/tree/main/References>.
- [11] Wikipedia foundation. “Windows file explorer.” (2022), [Online]. Available: https://en.wikipedia.org/wiki/Windows_File_Explorer (visited on 06/10/2022).

- [12] Basler AG. “Pylon 6.0.1 camera software suite windows.” (2022), [Online]. Available: <https://www.baslerweb.com/en/sales-support/downloads/software-downloads/pylon-6-0-1-windows/> (visited on 03/29/2022).
- [13] —, “How to build basler pylon c++ applications with free microsoft visual studio.” (2017), [Online]. Available: https://www.baslerweb.com/fp-1508418766/media/downloads/documents/application_notes/AW00064406000_Application_Note_Build_pylon_C_Apps_with_Free_Visual_Studio_IDE.pdf (visited on 03/29/2022).
- [14] —, “Pylon sdk samples manual.” (2021), [Online]. Available: https://www.baslerweb.com/fp-1615186793/media/downloads/documents/users_manuals/AW00148804000_pylon_SDK_Samples_Manual.pdf (visited on 03/29/2022).
- [15] A. Rosebrock. “Image stitching with opencv and python.” (2018), [Online]. Available: <https://pyimagesearch.com/2018/12/17/image-stitching-with-opencv-and-python/> (visited on 06/10/2022).
- [16] N. Lang. “Using convolutional neural network for image classification.” (2021), [Online]. Available: <https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4> (visited on 06/12/2022).
- [17] E. J. Broeders and E. H. Skov, “Foraminifera research, imaging and detection application,” Dec. 2019.
- [18] M. Ibrahim, “An in-depth efficientnet tutorial using tensorflow — how to use efficientnet on a custom dataset,” 2021. [Online]. Available: <https://towardsdatascience.com/an-in-depth-efficientnet-tutorial-using-tensorflow-how-to-use-efficientnet-on-a-custom-dataset-1cab0997f65c> (visited on 06/14/2022).
- [19] “Warning:tensorflow:your input ran out of data; interrupting training.” (), [Online]. Available: <https://github.com/fizyr/keras-retinanet/issues/1449> (visited on 06/12/2022).
- [20] V. J., “Tutorial on using keras flow_from_directory and generators,” 2018. [Online]. Available: <https://vijayabhaskar96.medium.com/tutorial-image-classification-with-keras-flow-from-directory-and-generators-95f75ebe5720> (visited on 06/14/2022).
- [21] A. Bhandari, “Image augmentation on the fly using keras imagedatagenerator!,” 2020. [Online]. Available: <https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/> (visited on 06/14/2022).
- [22] Genmitsu Industries LLC. “Original genmitsu 3018-pro.” (2022), [Online]. Available: <https://genmitsu.com/> (visited on 03/29/2022).

- [23] Desktop CNC Supply. “Woodpecker cnc control board v3.4.” (2022), [Online]. Available: <https://www.3018nc.com/product/3018-pro-cnc-router-control-board/> (visited on 03/29/2022).